PATENT APPLICATION

FOR

SYSTEM AND METHOD FOR SINGLE SESSION SIGN-ON

BY:

LAWRENCE R. MILLER, NEW YORK, NY &
MARTIN J. TRENHOLM, LONDON, UK

SYSTEM AND METHOD FOR SINGLE SESSION SIGN-ON

This application claims priority to U.S. Provisional Patent Application Ser. No. 60/338,359, filed December 4, 2001, entitled "SYSTEM AND METHOD FOR SINGLE SESSION SIGN-ON", the disclosure of which is incorporated herein by reference.

BACKGROUND

1. Field of the Invention

The present invention relates to authentication or credentials for access control of protected resources, and more particularly to the use of credentials or authentication granted by one system as a basis for granting credentials or authentication on another system.

2. Description of the Related Art

As known in the art, it is possible to have session credentials to control or limit access to protected resources. In a networked system, this technique is commonly used when a client computer attempts to gain access to protected resources that are held or accessible through a server. These credentials or authentication are typically granted to the client for the duration of a session. The session may be defined by the length of time that a browser application on the client computer is open, or it may be defined by the shorter of a specific period of time, and the length of time that the browser application is open. A session may also last for a longer time than the browser application is open.

Once the session is over, the credential or authentication is no longer valid and the client user must re-establish their credentials or authentication in order for them to again have access to the protected resources of the server.

A problem arises when the client wants access to protected resources on different servers of a system during the same session. Without some mechanism for sharing of

credentials or authentication between the servers, the client user must establish credentials with each server. To overcome this problem, single sign-on systems have been developed. While these single sign-on systems eliminate most or all of the necessity for a client user to authenticate on each system, they do not readily scale or bridge across different systems. One technique for bridging across different systems is to have a shared vault for authentication or credentials that is available to both systems. However, this approach requires a great deal of coordination between the systems, and necessarily requires some cross-system access.

Another approach is to have some form of shared secret keys or set of public keys used by the two systems, which allows one system to prove its' identity to the other by encrypting or signing a request and passing it through the client browser to the second system (typically this is done through a "cooked URL" or CURL.)

What is needed is a method and system to support cross-system authentication and credentialing, while maintaining the advantages of single system authentication and credentialing.

SUMMARY OF THE INVENTION

In one embodiment, the invention provides a method and system for validating credentials by determining at a first system that a client does not have a valid session credential for the first system, then retrieving at the first system, information from a session token (e.g. a cookie) held by the client, which corresponds to a possible session credential for the second system. At least some of the information from the session token is presented to the second system, and the second system determines whether the client has a valid session credential.

In another embodiment, the invention provides a method and system for establishing session credentials by determining that a client does not have a valid session credential for a first or a second system. In this embodiment, the system sends a log in page from the first system to the client, and receives log in information from the client. The system sends from the first system to the second system, the log in information, and receives, at the first system, information corresponding to a session credential for the second system, the session credential granted by the second system based at least in part on the log in information.

In another embodiment, the invention provides a method and system for establishing session credentials by determining that a client does not have a valid session credential for a first or a second system. In this embodiment, the system sends a log in page from the second system to the client, and receives log in information from the client. The system sends from the second system to the first system, information corresponding to a session credential for the second system, where the session credential granted by the second system is based at least in part on the log in information, and grants a session credential for the first system.

The foregoing specific aspects and advantages of the invention are illustrative of those which can be achieved by the present invention and are not intended to be exhaustive or limiting of the possible advantages that can be realized. Thus, the aspects and advantages of this invention will be apparent from the description herein or can be learned from practicing the invention, both as embodied herein and as modified in view of any variations that may be apparent to those skilled in the art. Accordingly the present

invention resides in the novel parts, constructions, arrangements, combinations and improvements herein shown and described.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing features and other aspects of the invention are explained in the following description taken in conjunction with the accompanying figures wherein:

FIG. 1 illustrates elements of a system according to one embodiment of the invention;

FIG. 2 illustrates steps in a method according to one embodiment of the invention;

FIG. 3 illustrates steps in a method according to one embodiment of the invention;

FIG. 4 illustrates steps in a method according to one embodiment of the invention; and

FIG. 5 illustrates steps in a method according to one embodiment of the invention.

It is understood that the drawings are for illustration only and are not limiting.

DETAILED DESCRIPTION OF THE DRAWINGS

Referring to FIG. 1 as an example, overall system 100 of the invention includes at least two separate systems 102, 104, each system with some form of protected resource. Overall system 100 also includes at least one individual client, with two clients illustrated as 106, 108 and a network 110 such as the Internet. Systems 102, 104 and clients 106, 108 may be individual computers, or networked computers. Although illustrated for only one computer, typically, each of the computers of systems 102, 104 and clients 106, 108 includes a central processor 112, memory 114, input and output devices 116, fixed storage media 118 and removable storage media 120.

Clients 106, 108 run operating system software and application software. In one embodiment, with network 110 used to connect systems 102, 104 and clients 106, 108,

Internet browser software, such as INTERNET EXPLORER or NETSCAPE are also run by clients 106, 108. Similarly, systems 102, 104 also run operating system software and application software. As indicated above, systems 102, 104 also include protected resources, frequently in the form of data stored as databases, and to support those resources, systems 102, 104 run server software and other software commonly associated with servers. To support the web sites that provide access to the protected resources, systems 102, 104 also run web server applications, such as: NETFUSION, EPICENTRIC, VIGNETTE, PEOPLESOFT, BEA WEBLOGIC PORTAL, or custom-developed enterprise applications such as the MorganMarkets website at JPMorgan, the JPMorgan Express Online application, etc.).

The protected resources of systems 102, 104 may include sensitive business information that is restricted to particular individuals or groups, or the protected resources may be subscription or pay-per-use. Credentials are also important for personalization. The protected resources of systems 102, 104 are stored on individual or multiple servers, which are not illustrated. Users of clients 106, 108 will want access to the protected resources of systems 102, 104, but to protect the resources of systems 102, 104, users or clients 106, 108 are first authenticated before they can gain access.

The authentication process usually includes a log in process where the user enters a user name and password. That user name and password is checked against a database and if valid, the user is allowed access to the protected resources.

Examples of commonly known authentication and credentialing software packages used by systems 102, 104 include GETACCESS and TRUEPASS by Entrust,

SITEMINDER by Netegrity, and IBM Policy Director. However, these software packages do not readily support cross-system authentication and credentialing.

Internet Browser Cookies

As part of the network and application protocols used by systems 102, 104, and clients 106, 108, it is common for cookies to be passed between systems 102, 104 and clients 106, 108. Because cookies and their uses can be an aspect of some embodiments of the invention, it is helpful to spend some time generally explaining what cookies are and how they work.

A cookie is a small piece of data that consists of a text-only string. It has provisions to include the domain, path, lifetime and value of a variable that the website (*e.g.*, systems 102, 104) sets. A cookie is an HTTP header that is typically sent from a server to a client and then may be sent from the client back to the server. Accordingly, some knowledge of HTTP, which can be found in RFC 2109, is helpful.

A cookie may contain six (6) parameters that can be passed. These are: 1) the name of the cookie; 2) the value of the cookie; 3) the expiration date of the cookie; 4) the path the cookie is valid for; 5) the domain the cookie is valid for; and 6) the need for a secure connection to exist to use the cookie. Of these six parameters, two parameters (the name and its value) are mandatory. The other four parameters are set either explicitly or by default.

An example of a cookie that might be sent from a server (*e.g.,* system 102, 104) to a client (*e.g.,* 106, 108) is:

Content-type: text/html

Set-Cookie: foo=bar; path=/promo; domain=www.myserver.com; expires Mon, 09-Dec-2002 13:46:00 GMT

The *name* and *value* parameters of a cookie are mandatory and are set by simply pairing them as in *name=value*. In the example above, the *name* parameter is foo, and the *value* is bar.

The *path* parameter sets the URL path the cookie is valid within. If there is no *path* parameter, the value defaults to the URL path of the document creating the cookie. Regardless of whether the server explicitly sets the *path* parameter, or the parameter is set by default, any web pages outside the *path* cannot read or use the cookie. The *path* parameter can have significant security and privacy implications and helps to ensure that cookies are not readily available except to the intended servers. In the example above, the *path* is /promo, and the cookie is only valid for web pages or documents on that path.

The *domain* parameter sets the domain that is allowed to access the cookie, and a server issuing a cookie must be a member of the domain that it tries to set in the cookie. Unless explicitly set, the *domain* parameter defaults to the full domain of the web page or document that sets the cookie. As examples, a server in the domain www.myserver.com cannot set a cookie for the domain .yourserver.com . However, a server in the domain www.yourserver.myserver.com can set a cookie for the domain .myserver.com . As discussed in greater detail below, this is important for some aspects of the invention. In the example above, the *domain* parameter is www.myserver.com .

The *expires* parameter determines the length of time the cookie is valid. If the server does not explicitly set the *expires* parameter, it defaults to the end of session.

Depending on the particular browser, this normally means that the cookie does not remain in any form of data storage after the browser session is complete, and for most browsers, it means that the cookie is held primarily or entirely within volatile memory and as soon as the browser application closes on the client, the cookie is forever lost. For most browsers (including NETSCAPE and INTERNET EXPLORER), setting the *expires* parameter causes the browser to store the cookie on disk, not only to hold it in volatile memory. In the example above, the cookie expires on Monday, December 9, 2002 at 13:46:00 GMT.

Depending on whether or not there is an *expires* parameter for a cookie, and the date of that parameter, it will be retained only in the volatile memory of clients 106, 108 and within memory allocated to the browser while the browser is running, or the browser may write the cookie to non-volatile storage or disk so that it is available even after the browser application is closed or stopped.

Cookies provide a way for a server to maintain state using HTTP, which is otherwise a stateless protocol, thereby avoiding the need for a client user to continuously re-identify themselves to a server, or authenticate themselves so as to gain access to protected resources of the server. For example, when a client user initially connects over the Internet or an Intranet to a server that has protected resources, and the client computer is using a browser application running on the client computer, the user may be asked to authenticate themselves through a log on page. The server is able to keep track of which users have previously logged on or authenticated themselves by checking for a cookie on the client browser. If the server knows what cookies have been set on clients and has a way to verify that a cookie returned by a client is valid, then the server can be reasonably

assured that if a client returns a valid cookie, the client can safely be granted further or continued access to the protected resources. There are many ways for a server to ensure that a cookie is valid and that the intended user is using the client computer. One such way is to encode, encrypt or hash the cookie value and to set the *expires* parameter to a short period of time, such as a few minutes. Then, as the user accesses different web pages, the server periodically updates the *expires* parameter of the cookie. This has the effect of maintaining a valid cookie and avoiding the need for the user to re-authenticate themselves, but at the same time helps to ensure that a different user of the client computer will not be able to access the servers' protected resources should the first and authenticated user happen to walk away from the client computer without terminating the browser application or logging out. However, most systems do not periodically reset the cookie to update the *expires* parameter. In most systems, the cookie is set once at session startup, either without specifying an *expires* parameter (in which case the cookie will only last the lifetime the browser is open) or specifying an *expires* parameter longer than the maximum lifetime of the session. The actual session expiration is typically handled by the server, which maintains a session record for the user, and includes total session lifetime, activity timeout information, and other session state.

The description of cookies that is provided above is not intended to be full or complete, but is provided to assist those of ordinary skill in understanding certain aspects of the invention. There are many sources of information on cookies, one of which can be found at http://www.cookiecentral.com/faq .

An Example Method

Referring now to FIGs. 1 and 2, a method of an embodiment of the invention begins at step 202 when a client (e.g., 106, 108) attempts to access a protected resource on system 1 (102).

At step 204, system 1 (102) determines whether the client has a valid single sign-on (SSO) session.

If the client has a valid SSO session, then at step 206, the client is granted access to the protected resource(s) of system 1 (102), and the method ends.

If, at step 204, it is determined that the client does not have a valid SSO session, then at step 208, system 1 (102) retrieves an SSO session token from the client. The token corresponds to a possible SSO session that the client has with another system (104). When the method of the invention is used with a web based application and browser, the token is the same as or similar to a cookie. When the method of the invention is used with systems other than the Internet and web based applications, the token is a piece of data or information that provides authentication or credentials of the client with system 2.

At step 210, after retrieving the token from the client, system 1 (102) sends the token or information extracted from the token to system 2 (104) as a request. In this step, system 1 (102) impersonates the client to system 2 (104).

At step 212, system 2 (104) receives the token or information extracted from the token.

At step 214, system 2 (104) determines whether the client (108) has a valid SSO session with system 2.

If at step 214, system 2 determines that the client has a valid SSO session, then at step 216, that information is communicated to system 1 (which is impersonating the client), and system 1 grants access to the client based on the clients' SSO session with system 2.

At step 218, the client's SSO session credentials with system 2 are periodically renewed. This renewal may be performed by system 1, system 2 or the client.

At step 220, the client has access to the protected resources of system 1 (102), as well as an SSO session with system 2 (104).

If at step 214, system 2 determines that the client does not have a valid SSO session, then at step 222, the client is provided an opportunity to log in. A previously valid session may become invalid if a timer has expired. In this case, if there is a cryptographic key associated with the cookie or token the key may be valid, but the cookie or token may have expired. One embodiment of the steps for client log in that are summarized at step 222 of FIG. 2 are more fully illustrated in FIG. 3 and described below.

Referring now to FIGs. 1 and 3, at step 300, system 2 (104) returns a redirect code to indicate that the client's session with system 2 is not valid. System 2 (104) sends this redirect code to system 1 (102).

At step 302, after receiving the redirect code, system 1 (102) directs the client to system 2 (104) in such a way that the system 2 log in server will redirect the client back to the system 1 log in page after authentication. In one embodiment this is done using a URL, such as https://www.yourserver.com/login?from=www.myserver.com. This

example will redirect the client to www.yourserver.com and tell the client to go back to www.myserver.com. There are other ways this can be written.

At step 304, the client receives the direction from system 1 and is redirected to the system 2 log in server.

At step 306, system 2 sends a log in page for display on the client browser. The log in page may be for system 1, system 2 or a custom page.

At step 308, the client user enters their name and password, or provides some other form of identification or authentication, such as a SECURID card or token available from RSA, biometrics, smartcard, etc.

At steps 310 and 312, system 2 checks the validity of the name and password or authentication of the client user.

If the name and password or authentication of the client user is valid, then at step 314, system 2 places a cookie or session token on the client browser. Then, at step 316, system 2 redirects the client back to system 1 (step 202 of FIG. 2). On this subsequent attempt of the client to access the protected resources of system 1, beginning at step 202 of FIG. 2, system 2 will find a valid SSO session at step 214, and system 1 can then grant the client access to the protected resources at step 216.

If at step 312, the name and password or authentication of the client is not valid, then at step 318, system 2 determines whether the client is allowed another attempt to authenticate, and if so, returns to step 306. Otherwise, the client is denied access at step 320.

Referring to FIG. 4, another embodiment of a log in method is illustrated.

At step 402, system 1 generates a log in page and sends the log in page to the client browser. Although sent by system 1, the log in page corresponds to system 2.

At step 404, the client browser displays the log in page, and at step 406, the client user, or an automated process, returns the required authentication credentials to system 1.

At step 408, system 1 collects the authentication credentials from the client browser and presents them to system 2, acting as the client.

At steps 410, 412, system 2 receives the credentials and authenticates the user.

If at step 412, system 2 determines that the user is valid, then at step 414, system 2 grants or validates the session credentials and returns them to system 1.

At step 416, based on the credentials granted by system 2, system 1 also generates credentials for the client (system 1 credentials) and sends both the system 1 and system 2 credentials to the client, thereby granting the client access to systems 1 and 2.

If at step 412, system 2 determines that the user is not valid, then at step 418, system 2 sends a reject message to system 1, and at step 420, system 1 displays an authentication error, and loops to step 402.

Specific Examples

Example 1. In this example, the two servers are in the same domain (per FIG. 2). The protocol in the example uses https, although it could be http.

Client has a previously established session with a server called "app2.jpmorgan.com" (system 2), and has session credentials from this system stored in the browser. The session credentials were stored by the browser due to app2.jpmorgan.com sending the following header to the client in response to the client's

initial log-in to app2.jpmorgan.com: Set-Cookie: sso2cookie=2938ryfhs8dsjdgfas832fdjdijhHyGg; path=/; domain=.jpmorgan.com .

Client attempts to access the URL "https://app1.jpmorgan.com/" (system 1). Server app1.jpmorgan.com checks the user's HTTP headers for a session cookie named "sso1cookie", which is the name of the session cookie used by app1.jpmorgan.com's single sign-on system. There is no valid session cookie.

Server app1.jpmorgan.com (system 1) checks the user's HTTP headers for a session cookie named "sso2cookie", which is the name of the session cookie used by app2.jpmorgan.com's (system 2) single sign-on system. It finds that there is a cookie "sso2cookie" with a value "2938ryfhs8dsjdgfas832fdjdijhHyGg".

Server app1.jpmorgan.com (system 1) cannot by itself determine if this cookie corresponds to a valid session.

Server app1.jpmorgan.com sends an HTTP GET request to the URL "https://app2.jpmorgan.com/checkSession", and includes the cookie "sso2cookie=2938ryfhs8dsjdgfas832fdjdijhHyGg" in the HTTP headers for the request.

Server app2.jpmorgan.com (system 2) receives the request, and extracts the cookie sso2cookie from the request headers.

Server app2.jpmorgan.com checks the value of sso2cookie and determines that "2938ryfhs8dsjdgfas832fdjdijhHyGg" represents a valid session for the user named "username".

Server app2.jpmorgan.com (system 2) generates an HTTP response with response code 200, of MIME type "text/plain" and with a body of "username", and returns this as the response to the request from app1.jpmorgan.com (system 1).

Server app1.jpmorgan.com (system 1) checks the response from app2.jpmorgan.com (system 2). The response code is 200, which indicates a valid response, and the body of the response is "username" which tells app1.jpmorgan.com the user ID of the user attempting to access the system.

Server app1.jpmorgan.com generates an authenticated session for user "username" on its own system, and generates a session credential corresponding to this user session of "243879h43908gjw55ksuywel9". Server app1.jpmorgan.com returns a response to the client with a status code of 200 containing the content corresponding to the URL "https://app1.jpmorgan.com", personalized for user "username" and displaying only content that user is allowed to see. In the response, it adds the header: Set-Cookie: sso1cookie=243879h43908gjw55ksuywel9; path=/; domain=.jpmorgan.com .

Now that app1.jpmorgan.com has created a session for the user, subsequent requests will be accepted based on the presence of the cookie named "sso1cookie" in the request headers sent from the client.

Example 2. In another example, the two servers are in different domains (Referring now to FIG. 5):

Client has a previously established session with a server called www.chase.com (system 2), and has session credentials from this system stored in the browser. The session credentials were stored by the browser due to app.chase.com sending the following header to the client in response to the client's initial log-in to www.chase.com: Set-Cookie: chasesso=2938ryfhs8dsjdgfas832fdjdijhHyGg; path=/; domain=.chase.com

At step 502, client attempts to access the URL "https://www.jpmorgan.com/"

At step 504, server www.jpmorgan.com (system 1) checks the user's HTTP headers for a session cookie named "jpmsso", which is the name of the session cookie used by www.jpmorgan.com's single sign-on system.

If there is no valid session and no valid cookie, then at step 506, server www.jpmorgan.com (system 1) sends an HTTP response code of 302 ("redirect") to the client browser, with a redirection URL of "https://www.chase.com/getCredentials?from=www.jpmorgan.com"

At step 508, the client browser receives the response from www.jpmorgan.com and makes an HTTP GET request to the URL "https://www.chase.com/getCredentials?from=www.jpmorgan.com" .

At step 510, the server www.chase.com (system 2) verifies that www.jpmorgan.com is a site with which session credentials may be shared, and sends an HTTP response code 302 ("redirect") to the client browser, with a redirection URL of "https://www.jpmorgan.com/login?chasesso=2938ryfhs8dsjdgfas832fdjdijhHyGg". Note that the redirection URL has as an argument the SSO credential for the user on the www.chase.com server.

At step 512, server www.jpmorgan.com (system 1) sends an HTTP GET request to the URL "https://www.chase.com/checkSession", and includes the cookie "chasesso=2938ryfhs8dsjdgfas832fdjdijhHyGg" in the HTTP headers for the request.

At step 514, server www.chase.com (system 2) receives the request, and extracts the cookie chasesso from the request headers.

At step 516, server www.chase.com checks the value of chasesso and determines that "2938ryfhs8dsjdgfas832fdjdijhHyGg" represents a valid session for the user named "username".

If there is a valid session for the user named "username", then server www.chase.com (system 2) generates an HTTP response with response code 200, of MIME type "text/plain" and with a body of "username", and returns this as the response to the request from www.jpmorgan.com.

At step 518, server www.jpmorgan.com (system 1) checks the response from www.chase.com. The response code is 200, which indicates a valid response, and the body of the response is "username" which tells www.jpmorgan.com the user ID of the user attempting to access the system. Server www.jpmorgan.com (system 1) generates an authenticated session for user "username" on its own system, and generates a session credential corresponding to this user session of "243879h43908gjw55ksuywel9".

At step 520, server www.jpmorgan.com (system 1) returns a response to the client with a status code of 200 containing the content corresponding to the URL "https://www.jpmorgan.com", personalized for user "username" and displaying only content that user is allowed to see. In the response, it adds the header: Set-Cookie: jpmsso=243879h43908gjw55ksuywel9; path=/; domain=.jpmorgan.com .

Now that www.jpmorgan.com (system 1) has created a session for the user, subsequent requests will be accepted based on the presence of the cookie named "jpmsso" in the request headers sent from the client.

Although illustrative embodiments have been described herein in detail, it should be noted and will be appreciated by those skilled in the art that numerous variations may

be made within the scope of this invention without departing from the principle of this invention and without sacrificing its chief advantages. As examples of alternatives, some of the steps that are illustrated and described above may be omitted, or additional steps may be added.

The description provided above uses the Internet, browser applications and cookies. However, there is no intention to limit the invention to implementation using only the Internet, browser applications and cookies. The primary aspects are that session credentials that are held by one system (*e.g.*, system 2) are used to establish or grant session credentials on another system (*e.g.*, system 1), and the session credentials of system 2 are such that they are not directly available to or accessible by system 1, but held by the client as part of a session token or "cookie", and the session token information can be extracted by system 1 and then validated or authenticated with system 2.

Unless otherwise specifically stated, the terms and expressions have been used herein as terms of description and not terms of limitation. There is no intention to use the terms or expressions to exclude any equivalents of features shown and described or portions thereof and this invention should be defined in accordance with the claims that follow.